CS156 f25

lab / team02

Canvas link: https://ucsb.instructure.com/courses/27687/assignments/381964

team02—Team Project: React CRUD frontend

What are we learning in team02

In this team project, our starter code has a frontend and backend, and we are focusing only on the frontend; you will likely not have to make any modifications to the backend (other than copying your backend code from team01).

We are focusing on learning these new React concepts:

- Adding new items to the navbar
- Adding components for new placeholder pages
- Adding routes in `App.js` to link urls to pages
- Creating a component that uses the `OurTable` component to create a table
- Changing the placeholder page so that it loads data using the `useBackend` hook to fetch data
- Using the `ButtonColumn` feature of `OurTable` to add a delete button
- Using `npm test` to run unit tests
- Using `npm run coverage` to run test coverage
- Using `npx stryker run` to run mutation coverage
- Using `npm run storybook` to visualize frontend components in isolation

Repos for team02

Here are the links to the artifacts for team02:

Repo	Kanban Board	GH Pages	Dokku	Dokku QA	Slack
team02-f25-01	team02-f25-01	team02-f25-01	dokku	dokku-qa	slack

Repo	Kanban Board	GH Pages	Dokku	Dokku QA	Slack
team02-f25-02	team02-f25-02	team02-f25-02	dokku	dokku-qa	slack
team02-f25-03	team02-f25-03	team02-f25-03	dokku	dokku-qa	slack
team02-f25-04	team02-f25-04	team02-f25-04	dokku	dokku-qa	slack
team02-f25-05	team02-f25-05	team02-f25-05	dokku	dokku-qa	slack
team02-f25-06	team02-f25-06	team02-f25-06	dokku	dokku-qa	slack
team02-f25-07	team02-f25-07	team02-f25-07	dokku	dokku-qa	slack
team02-f25-08	team02-f25-08	team02-f25-08	dokku	dokku-qa	slack
team02-f25-09	team02-f25-09	team02-f25-09	dokku	dokku-qa	slack
team02-f25-10	team02-f25-10	team02-f25-10	dokku	dokku-qa	slack
team02-f25-11	team02-f25-11	team02-f25-11	dokku	dokku-qa	slack
team02-f25-12	team02-f25-12	team02-f25-12	dokku	dokku-qa	slack
team02-f25-13	team02-f25-13	team02-f25-13	dokku	dokku-qa	slack
team02-f25-14	team02-f25-14	team02-f25-14	dokku	dokku-qa	slack
team02-f25-15	team02-f25-15	team02-f25-15	dokku	dokku-qa	slack

Repo	Kanban Board	GH Pages	Dokku	Dokku QA	Slack
team02-f25-16	team02-f25-16	team02-f25-16	dokku	dokku-qa	slack

For team02, the Kanban board is populated by the staff before you start using the Github Actions workflow `99-team02.yml`; if your Kanban is not yet populated, please check in with the staff.

Dokku prod deployments

The prod deployments (i.e. the ones on dokku with the name `team02`) should track the `main` branch. **You will need to manually redeploy** after you merge each pull request, with these commands (adapted for your team; the example shows team f25-01):

```
dokku git:sync team02 https://github.com/ucsb-cs156-f25/team02-f25-01 main
dokku ps:rebuild team02
```

Work on your own laptop, not CSIL

You should be working with a Java and Node (Javascript) setup on your own laptop by this point, not on CSIL.

For advice on what to install, see:

- https://ucsb-cs156.github.io/f25/info/software.html

If this presents a difficulty, please contact the instructor ASAP so that some arrangement can be made for your situation.

A few words about Chromatic

This quarter, f25 is the **first time** that we are trying to incoporate Chromatic visual difference testing into the course.

Accordingly, we may run into **unexpected issues**. Sometimes the only way to know how a technology will work when you turn 96 students loose on it in a course is to **try it, and see what happens**.

The dropdown box below will contain information about issues/problems we run into using Chromatic, and it **may be updated periodically** as we run into additional issues. Please check it

periodically if you run into Chromatic issues while working on this lab.

▼

Yellow Circle on Chromatic Checks (Github Actions)

If you find that one of the Github Actions related to Chromatic is not finishing (i.e. stays as a permanent yellow circle), consult this page for hints on what to do:

- https://ucsb-cs156.github.io/topics/chromatic/chromatic_yellow_circle.html

What are we doing in team02?

In this team project, you'll continue the work you started in team02.

The starter code has examples of:

- Implementing backend CRUD operations for two database tables (this is exactly like team01)
- Implementing a user-friendly frontend for the CRUD operations for these two database tables using Javascript and React (this is the new part)

It also has an example of temporary placeholder pages for CRUD operations, which is an intermediate step towards the full CRUD frontend code.

Basically, the task of team02 is to use this example code to build frontends for the backends you built in team01).

Understanding what you are building

The best way to understand what you are building is to play with a deployment of the starter code.

You can try out an example of the user-friendly CRUD frontend at the link below; it's a deployment of the starter code repo https://github.com/ucsb-cs156-f25/STARTER-team02. If you login with your UCSB email, you should have admin access (provided you are a registered student this quarter), and you should be able to do CRUD operations on two database tables, `Restaurants` and `UCSBDates` (more information on this later in this project writeup).

- https://team02.dokku-00.cs.ucsb.edu/

I suggest you take a moment to experiment with various pages of that app. Here are some things to explore:

- When not logged in, you'll have no access to the CRUD operations for the app. But when you login, you should see two new items on the menu for `Restaurants` and `UCSBDates`.
- All registered students should have admin access when logging in with their UCSB email. This will allow you to add new records and edit existing ones.
- If you have another Google email, you can log in with that; you should see that you have read only access to the two database tables.
- You'll also see a menu item called `Placeholder`. This is example code for making a set of "placeholder pages", which are a useful intermediate step when building a new set of pages for the table (more on that later).
- The initial page that comes up at the `Restaurants` or `UCSBDates` link is called the *Index Page*. It lists all of the records in the table. Note that you can sort by any column by clicking in the header.
- There is a link at the top of each page that takes you to the *Create Page* where you can create new database records. Try it out.
- When you create a new record, you are redirected back to the *Index Page*.
- There are also buttons that allow you to Delete or Edit a record. Try them as well. Note that the Delete button leaves you on the Index page, while the Edit button takes you to a separate Edit page. Note that the URL changes.

What you'll be doing (as a team) is adding additional menu items alongside `Restaurants` and `UCSBDates`, providing similar CRUD operations for four, five or six of these tables (the sames ones from team02):

- `UCSBDiningCommonsMenuItem`
- `UCSBOrganization`
- `RecommendationRequest`
- `MenuItemReview`
- `HelpRequest`
- `Articles`

You'll reuse the backend files you built for team02; copying them over is one of the first steps in the lab. Then you'll build a frontend in Javascript using the React framework.

Understanding the Starter Code

A React frontend is built using *Components*. In our code base, each React component is in a separate file under one of these two directories:

- `frontend/src/main/pages`: each of these components provides a single web page of the application. These components are registered in a file called `frontend/src/App.js` which maps frontend routes (i.e. URLs) to the React Component that provides that page.
- `frontend/src/main/components`: each of these components is a portion of a page.

So the files under `pages` and `components` are technically both "React Components"; the page files are a "special case" of components that are mapped to routes in `App.js`.

Typically, we try to factor out as much of the user interface code as possble into the component files, especially any code that can be reused.

Page files typically are responsible for getting and or posting data to/from backend APIs and communicating with components that make up the page.

Data can be passed into components through `props`, the parameters that each component takes. By convention, these are passed in a Javascript object, which is why you see `{}` around the names of the parameters to React Components.

Data is typically returned from components via "callback functions" that are passed into the components as part of their props.

Finally, note that under both `frontend/src/main/components` and `frontend/src/main/pages`, you'll see that the code is sometimes organized into subdirectories(folders); this is at the discretion of the coder, but it does make finding things easier, so it is a recommended practice. Consistent naming conventions also help keep the code organized.

Tests and Stories

For each React component, whether it's under `components` or `pages`, we have tests and stories.

- The tests are automated unit tests, similar to the JUnit tests we wrote for our Java code. Just as we used JUnit for Java, for our React code we use a framework called Jest, along with tools for test coverage, and mutation testing (Stryker).
- The stories are for a tool called Storybook. Storybook allows both interactive testing (i.e. human testing) of React Components in isolation from other components; think of it as "live unit testing". It also provides a way to document our catalog of Components. Storybook fills the same niche for frontend programming that Swagger fills for backend programming.

As you work through the issues on the Kanban board for this project, you'll see that for each React component we develop, we are also going to implement tests and stories for each one.

You can experiment with the published storybook for the Starter code at this link:

- https://ucsb-cs156-f25.github.io/STARTER-team02/chromatic

Note that when you interact with pages in the app, they are connected to a real backend with a real database. By contrast, when you interact with the pages and components in the storybook, all of the interactions are "mocked":

- For `GET` requests, we load data from mock endpoints to test the component in isolation.
- For `POST,PUT,DELETE` requests, we log the calls using `console.log` and/or `window.alert` so that can see what the component would have sent to the backend.

We can set up multiple "stories" for each component with different values for the mocked GET calls, as well as different values of the "props"(parameters) passed into each React component.

This has several benefits:

- Writing the stories helps to identify and document the APIs and prop values on which our React components depend.
- It also gives us a way to interactively test the component under various conditions, similar to what we do with Unit tests, but with the advantage that we can see what's happening directly in our browser
- Finally, when bugs arise in the real application, we can use Storybook in combination with Swagger to determine whether the problem is in the backend, the frontend, or the communication between the two.

Assigning the database tables to your team

The tables you'll need to divide up among the team members are the same ones as in team02. You can assign team members to the same tables, or you can switch it up; that's up to you.

The six tables are:

- `UCSBDiningCommonsMenuItem`
- `UCSBOrganization`
- `RecommendationRequest`
- `MenuItemReview`

- `HelpRequest`
- `Articles`

I suggest you divide up these tables on your Slack channel first. There is also an issue on the Kanban board to update this in the `README.md` of your team02 repo.

The examples: Restaurant and UCSBDate

The two database tables in the starter code are these:

- `Restauarants`, which represents data about a restaurant; just a name and description, such as:

```
[
      {
          "id": 2,
           "name": "Cristino's Bakery",
           "description": "Takeout only.  It may look mostly like a bakery with Mexican pastries, but it also
      },

      {
          "id": 3,
           "name": "Freebirds",
           "description": "Burrito joint, and iconic Isla Vista location"
      },

      {
          "id": 4,
           "name": "Ca' Dario Cucina Italiana",
           "description": "White tablecloth Italian restaurant, with great pasta and pizza"
      }
]
```

- `UCSBDate`, which represents dates from the UCSB calendar.

  The UCSBDates table is intended as a place to store dates such as these The ideas is to store each date as a single record, like this:

```
[
  {
    "quarter": "20234",
    "name": "firstDayOfClasses",
    "datetime": "2023-09-28T00:00:00",
  },
  {
```

```
    "quarter": "20234",
    "name": "lastDayOfClasses",
    "datetime": "2023-12-08T00:00:00",
  },
  {
    "quarter": "20234",
    "name": "firstDayOfFinals",
    "datetime": "2023-12-09T00:00:00",
  }
]
```


▼ More details about this table

Quarters are represented by a five digit number where the first four digits are the year, and the final digit is either 1, 2, 3, or 4, for Winter, Spring, Summer and Fall, respectively. This may seem like a weird format, but it's the one used internally by systems like GOLD. The `yyyyq` format has the advantage that when you sort, the quarters come out in the right order. That would not be true if you sorted `['F22', 'W23', 'S23','F23', 'W24']` in lexicographic order; they would come out `['F22', 'F23', 'S23', 'W23', 'W24']` which is an order that makes no sense.

For reference only, here's a complete list of UCSBDates for Fall 2023. This list is in the original format used by the UCSB Developer API's Academic Quarter Calendar. Note that in the format, each record is for a single quarter, while in our table, each record represents a single date.


```
{
  "quarter": "20234",
  "qyy": "f23",
  "name": "FALL 2023",
  "category": "FALL",
  "academicYear": "2023-2024",
  "firstDayOfClasses": "2023-09-28T00:00:00",
  "lastDayOfClasses": "2023-12-08T00:00:00",
  "firstDayOfFinals": "2023-12-09T00:00:00",
  "lastDayOfFinals": "2023-12-15T00:00:00",
  "firstDayOfQuarter": "2023-09-24T00:00:00",
  "lastDayOfSchedule": "2023-12-15T00:00:00",
  "pass1Begin": "2023-05-15T09:00:00",
  "pass2Begin": "2023-05-22T08:30:00",
  "pass3Begin": "2023-09-11T09:00:00",
```

```
  "feeDeadline": "2023-09-15T00:00:00",
  "lastDayToAddUnderGrad": "2023-10-18T00:00:00",
  "lastDayToAddGrad": "2023-12-08T00:00:00",
  "lastDayThirdWeek": null
}
```


An aside about the placeholder pages

You may have noticed a menu item called `Placeholder` on the starter code deployment here (login and try it out if you haven't yet):

- https://team02.dokku-00.cs.ucsb.edu/

Normally, we would not want real users to see placeholder pages such as these in a real app; but there are times that we might want them to be in the `main` branch of our code so that we can start integrating a new feature early.

One way to resolve this is with the use of something called Feature Toggles. Feature toggles allow us to configure an app to run in different modes in different deployments, and are sometimes used to release a new feature in stages. It can be put into the `main` branch, but enabled in some developer and qa deployments and disabled in others. The important things to keep in mind are:

- The app should be tested with the feature both on and off before PRs are merged to `main`
- This allows a new feature to be incorporated into the `main` branch in stages until it is finally ready to show to customers.
- Sometimes, feature toggles can be on a user-by-user basis, so that a new feature can be introduced to some "early adopter" users that have a higher tolerance for things that are still in an early state; this allows early feedback about new features, which is a key part of the Agile software design philosophy.

Note that we are not using feature toggles in team02 (at least not as of this quarter, though it would be a nice thing to incorporate into a future version of the project.) But I wanted to be clear about how this "placeholder" code might be handled if this were a real-world code base, and not an exercise.

Understanding the issues on the Kanban board

The issues on your Kanban board can be grouped as follows:

- Project Scope issues (6 issues): These are one-time tasks, and should be divided among the team members in some reasonably equitable fashion, at the discretion of the team. Note that 'equitable' doesn't necessarily mean 'equal'; not all of these issues involve an equal amount of work. It's more a sense among the team members that the workload is being shared in a "fair" way, whatever that means to the members of your team.
  - Add table to README.md (table of who is doing which table)
  - Set up GitHub Pages
  - Set up Team prod deployment
  - Set up Team qa deployment
  - Adjust links in README.md (links to prod and qa deployments, etc.)
  - Submit for grading on Canvas (on behalf of team)
- Database Table Scope issues (48 issues, 8 per table): These are repeated for each database table
  - Backend Controller and Tests (copying from team01)
  - Fixtures (example data for testing)
  - Form Component (an HTML form for data entry)
  - Table Component (a HTML Table for visualizing lists of data)
  - Copy Placeholder pages; add to App.js/AppNavbar.js
  - Create Page (a page where admins can create new items in the table)
  - Edit Page (a page where admins can edit existing records)
  - Index Page (a page where any logged in user can list all items, and where admins can click to create, edit, or delete items)

Note that if you have fewer than 6 team members, you should go ahead and close the issues for database tables that your team is not implementing; you can also remove them from the Kanban board.

Copying your work into the starter code.

You'll start with the starter code from STARTER–`team02`, and then add to it the files you created in team01 to implement the database tables and backend CRUD operations

These will likely include, for each of the database tables you created in team01:

- A Java class for the `@Entity`
- A Java class for the `@Repository`

- The database migration file (`TableName.json`)
- A Java class for the Controller
- A Java class containing Controller tests

The first pull request you make should simply copy those files from your team01 repo into the appropriate directories.

I strongly encourage *one pull request per team member*, rather than one giant pull request for the entire team.

- Historically, when teams have tried the one "giant pull request" approach, it sometimes works, but *often a problem with one of the database tables holds up all the rest*.
- Generally speaking, small, focused PRs are easier to get green on CI, code review and get merged, vs. large sprawling ones.

General workflow for frontend code

To test the backend and frontend together:

- At top level of repo, run `mvn spring-boot:run` in one window (you'll need to configure your `.env` file with `GOOGLE_CLIENT_ID`, `GOOGLE_CLIENT_SECRET`, and `ADMIN_EMAILS`)
- In a separate terminal window, cd into `frontend`, type `nvm use 22.18.0` and then run `npm start`

Remember that the first time you do that, you must first do `npm ci`, like this:

```
nvm use 22.18.0
npm ci
npm start
```

Here are some other commands, all things you do in the `frontend` directory after first doing `nvm use 22.18.0` once in that session:

- Run tests locally: `npm test`.
- Run tests from one file locally: `npm test -- RestaurantsEditPage`
- Quickly test coverage locally: `npm run coverage`
- Check linting locally: `npx eslint --fix .`
- Check mutation coverage locally (slow): `npx stryker run`
- Check mutation coverage of single file (faster): `npx stryker run -m frontend/src/main/pages/Restaurants/RestaurantsEditPage.js`

- To run storybook locally: `npm run storybook`.

Adding the placeholders

When adding the placeholders, you'll be making changes to `App.js`, `AppNavbar.js`, and `AppNavbar.test.js`.

You may end up with merge conflicts, but they should be easy to resolve.

The fact that you'll end up with merge conflicts here is a feature, not a bug:it's helpful to get some experience with how to resolve those in a context where:

- The merge conflicts are real, not contrived
- They are straightforward to resolve

Having some experience in resolving these will help you later on when you encounter more complex merge conflicts.

Carrying out your issues

Now, take on each of the issues for your table on the Kanban board.

There are suggestions about how to proceed below.

Make a new branch each time, and do a separate pull request for each issue. In some cases, you may need to start your branch from a previous one.

As you finish each issue:

- Do a pull request, and ask for code reviews
- Check that the CI/CD tests are green. If they aren't fix that before starting the next issue.
- Once CI/CD is green, you do *not have to wait* for the code review and the PR to be merged before staring the next issue; you can just make a new branch off the previous branch and continue coding.
- However, we strongly encourage you to work with your team to try to get PRs merged as quickly as possible. Don't let them pile up.

Also, and I can't emphasize this enough: **keep each PR small**. We are trying to get you into that habit, because it will pay off in the project phase, and in real world practice.

When all of your issues are either complete, or at least have open pull requests, turn your attention to helping the team to get finished.

When done with your issues: check in with the team

- See if there are code reviews that you can help with
- See if anyone else on the team needs help with their tasks

What comes after team02

After team02, we have one more team assignment, where we focus on integration and end-to-end testing. After that, we start the legacy code projects.

Asking for help

If you need additional guidance, ask on the `#help-lecture-discussion` channel during class time, or on `#help-team02` outside of class time.

When you are done

When all branches are merged to main, all tasks on Kanban board in the done column, please submit on Canvas as you did for team01.

Videos

The following videos for team02 contain helpful information about carrying out the various steps.

Note that these are from an earlier quarter so if you find that you don't have access, let the instructor know; they may need to adjust the permissions.

Also, there may be slight differences between the assignment in earlier quarters and the assignment this quarter; in case of any discrepency, follow the instructions in this file and in the

issues on your Kanban board. You can also seek clarification on the channel.

Note that these videos refer to **team03** instead of **team02** (we eliminated one earlier assignment this quarter.)

Link to Video	Length	What it Covers
Intro to Front End, Part 1	26 minutes	Getting Started with the Frontend
Intro to Front End, Part 2	45 minutes	Debugging, Storybook
Adding Delete Button	18 minutes	Adding Delete Button to Table, and first look at Frontend Testing
FrontEnd Testing part 1	20 minutes	`npm run coverage`
FrontEnd Testing part 2	14 minutes	`npm run coverage` (continued)
FrontEnd Testing part 3	19 minutes	`eslint` and `stryker`

New Videos

The following videos for team02 contain helpful information about carrying out the various steps, and are *new* for Fall 2024.

Title	Link to Video	Length
team02/frontend, 01: Create personal dev deployment	https://youtu.be/mFv0cZJViqQ	17 minutes
team02/frontend, 02: Copy files for Backend CRUD API from team01 (previous project)	https://youtu.be/x_OaPMbPgMM	16 minutes
team02/frontend: 03: Create fixtures	https://youtu.be/u-jJ-ZWFpeE	30 minutes
team02/frontend: 04: Form Component plus tests (part 1)	https://youtu.be/clTcUGZlGq0	1 hour 15 minutes
team02/frontend: 05: Form Component plus tests (part 2, mutation testing)	https://youtu.be/v116IJyqj6k	35 minutes

Title	Link to Video	Length
team02/frontend: 06: Code Review and Merging PRs	https://youtu.be/Dh9RAXQcg5E	18 minutes
team02/frontend: 07: Table Component plus tests	https://youtu.be/kNJuBDpOstU	49 minutes
team02/frontend: 08: Copy Placeholders for pages; add to App.js/AppNavbar.js (Includes discussion of `git pull --rebase origin branch` and resolving merge conflicts during a rebase)	https://youtu.be/7DZ6LOvNAh0 UPDATED LINK!	42 minutes
team02/frontend: 09: Create Page plus tests and stories	https://youtu.be/gRAVdQLghgI	1 hour 2 minutes
team02/frontend: 10: Index Page plus tests and stories	https://youtu.be/Z4RCkLoQi0g	50 minutes
team02/frontend: 11: Edit Page plus tests and stories	https://youtu.be/2vyxAtskECU	1 hour 19 minutes

Total running time: 7 hours, 59 minutes

Staff information

▼

Information for staff is in this dropdown

Before releasing this lab to students, be sure the following tasks are done:

- Finalize starter code repo https://github.com/ucsb-cs156-f25/STARTER-team02
- Create team02 repos using the github actions in https://github.com/ucsb-cs156-f25/membership-scripts
  - public

    - team access admin
    - signed commits
- Create a Canvas assignment for team02; update the due dates and publish it
- Create projects for all of the groups. The best way is to create a Template that has precisely the setup you want (i.e. the columns, etc.) and then manually create the team kanban boards. There is not yet an API that will create the project with a precise configuration of columns, etc. so this is still the easiest way. Make sure the projects are named with the same exact names as the repos; this makes a later step easier.
- Then, update `_config.yml` for the website with the project numbers for team01. This causes the table of links in this lab to render properly.
- Then, in https://github.com/ucsb-cs156-f25/membership-scripts, there is a Github Action that will
    - Link the project to the repo
    - Set the permission for the project properly
- Double check that the kanban boards and repos are set up and have the correct permissions.
- Make sure the app https://team02.dokku-00.cs.ucsb.edu/ is up and running, and is sync'd with the starter code:

  i.e, on dokku-00 for example, do:

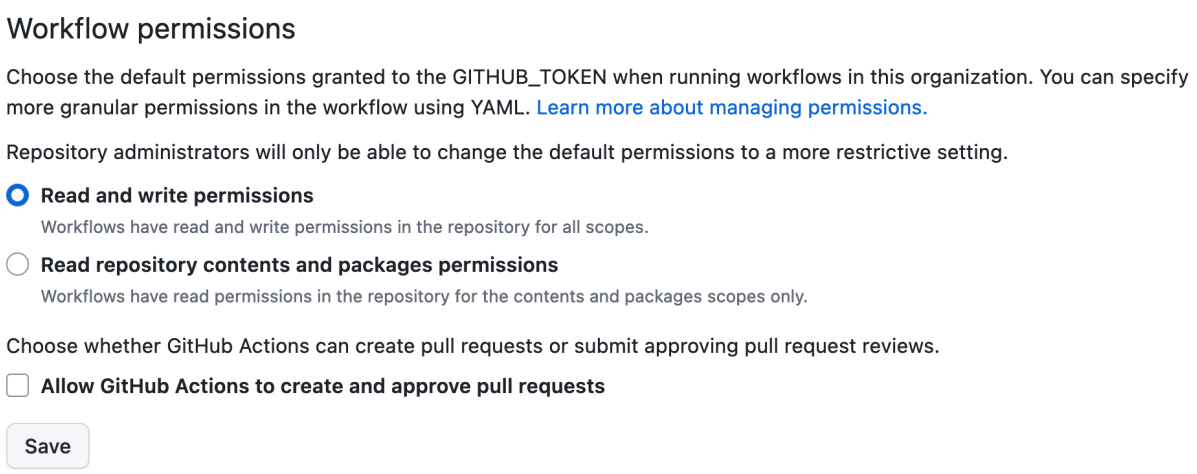
  ```
  dokku git:sync  https://github.com/ucsb-cs156-f25/STARTER- main
  dokku ps:rebuild
  </pre>
  ```

- Set up storybook (and the rest of the gh pages site): https://ucsb-cs156-f25.github.io/STARTER-team02/chromatic
- Proofread the instructions in this file, and request that the staff (TAs/LAs do also)
- Consider assigning at least one TA/LA (preferably the one with the least prior experience with the course) to complete the lab in it's entirety to debug the starter code and instructions

The next step was probably already done earlier, but just in case:

- Be sure that the organization settings are set like this, in, for example, https://github.com/organizations/ucsb-cs156-f25/settings/actions

This is needed so that the github actions scripts have write access to the directory.



This setting is probabaly also a good idea:

- Be sure that `#help-team02` is set up.
- Run the script to set up the Issues (action 99, see links below)
- Copy issues to the Kanban boards

Links for staff setup

Repo	Kanban Board	GH Pages	Action 99
team02-f25-01	team02-f25-01	team02-f25-01	99-team02.yml
team02-f25-02	team02-f25-02	team02-f25-02	99-team02.yml
team02-f25-03	team02-f25-03	team02-f25-03	99-team02.yml
team02-f25-04	team02-f25-04	team02-f25-04	99-team02.yml
team02-f25-05	team02-f25-05	team02-f25-05	99-team02.yml

Repo	Kanban Board	GH Pages	Action 99
team02-f25-06	team02-f25-06	team02-f25-06	99-team02.yml
team02-f25-07	team02-f25-07	team02-f25-07	99-team02.yml
team02-f25-08	team02-f25-08	team02-f25-08	99-team02.yml
team02-f25-09	team02-f25-09	team02-f25-09	99-team02.yml
team02-f25-10	team02-f25-10	team02-f25-10	99-team02.yml
team02-f25-11	team02-f25-11	team02-f25-11	99-team02.yml
team02-f25-12	team02-f25-12	team02-f25-12	99-team02.yml
team02-f25-13	team02-f25-13	team02-f25-13	99-team02.yml
team02-f25-14	team02-f25-14	team02-f25-14	99-team02.yml
team02-f25-15	team02-f25-15	team02-f25-15	99-team02.yml
team02-f25-16	team02-f25-16	team02-f25-16	99-team02.yml

If setting up starter code and Kanban boards for students

Clone all repos: change directory into a directory where you can clone all of the repos like this:

```
for team in {01..16}
do
  echo "Setting up code for $team"
  git clone git@github.com:ucsb-cs156-f25/team02-f25-${team}.git
done
```

Add starter remotes: use a loop like this to add the starter remotes:

```
for team in {01..16}
do
  echo "Adding starter remote for $team"
  cd team02-f25-${team}
  git remote add starter https://github.com/ucsb-cs156-f25/STARTER-team02
  cd ..
done
```

First time pull from starter: use a loop like this to pull in the starter code and push to main

```
for team in {01..16}
do
  echo "pull from starter for $team"
  cd team02-f25-${team}
  git pull starter main
  git push origin main
  cd ..
done
```

Subsequent pulls from starter: use a loop like this to pull in updates to the starter code:

```
for team in {01..16}
do
  echo "pull from starter for $team"
  cd team02-f25-${team}
  git checkout main
  git pull origin main
  git pull starter main
  git push origin main
  cd ..
done
```

Workflows to set up branch protection, etc

Once each repo has a main branch, consider running workflow 71 from the membership-scripts repo which sets up branch protection rules.